```
import numpy as np # linear algebra
import pandas as pd # data processing
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))


import warnings

warnings.filterwarnings('ignore')


data = '/content/car_evaluation.csv'

df = pd.read_csv(data, header=None)


# view dimensions of dataset

df.shape
```

```
(1728, 7)
```

```
# preview the dataset

df.head()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | vhigh | vhigh | 2 | 2 | small | low | unacc |
| 1 | vhigh | vhigh | 2 | 2 | small | med | unacc |
| 2 | vhigh | vhigh | 2 | 2 | small | high | unacc |
| 3 | vhigh | vhigh | 2 | 2 | med | low | unacc |
| 4 | vhigh | vhigh | 2 | 2 | med | med | unacc |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']


df.columns = col_names

col_names
```

```
['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
# let's again preview the dataset

df.head()
```

| | buying | maint | doors | persons | lug_boot | safety | class |
|---|---|---|---|---|---|---|---|
| 0 | vhigh | vhigh | 2 | 2 | small | low | unacc |
| 1 | vhigh | vhigh | 2 | 2 | small | med | unacc |
| 2 | vhigh | vhigh | 2 | 2 | small | high | unacc |
| 3 | vhigh | vhigh | 2 | 2 | med | low | unacc |
| 4 | vhigh | vhigh | 2 | 2 | med | med | unacc |

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1728 non-null   object
```

```
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

```
for col in col_names:

    print(df[col].value_counts())
```

```
buying
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
maint
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
doors
2        432
3        432
4        432
5more    432
Name: count, dtype: int64
persons
2       576
4       576
more    576
Name: count, dtype: int64
lug_boot
small    576
med      576
big      576
Name: count, dtype: int64
safety
low     576
med     576
high    576
Name: count, dtype: int64
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

```
df['class'].value_counts()
```

| class | count |
|---|---|
| unacc | 1210 |
| acc | 384 |
| good | 69 |
| vgood | 65 |

```
# check missing values in variables

df.isnull().sum()
```

| | 0 |
|---|---|
| buying | 0 |
| maint | 0 |
| doors | 0 |
| persons | 0 |
| lug_boot | 0 |
| safety | 0 |
| class | 0 |

```
X = df.drop(['class'], axis=1)

y = df['class']


# split data into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)


# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
((1157, 6), (571, 6))
```

```
# check data types in X_train

X_train.dtypes
```

| | 0 |
|---|---|
| buying | object |
| maint | object |
| doors | object |
| persons | object |
| lug_boot | object |
| safety | object |

```
X_train.head()
```

| | buying | maint | doors | persons | lug_boot | safety |
|---|---|---|---|---|---|---|
| 48 | vhigh | vhigh | 3 | more | med | low |
| 468 | high | vhigh | 3 | 4 | small | low |
| 155 | vhigh | high | 3 | more | small | high |
| 1721 | low | low | 5more | more | small | high |
| 1208 | med | low | 2 | more | small | high |

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl.metadata (8.0 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.3.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (2.1.4)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_enco
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encode
Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 2.5 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```
# import category encoders

import category_encoders as ce


# encode categorical variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)


X_train.head()
```

| | buying | maint | doors | persons | lug_boot | safety |
|---|---|---|---|---|---|---|
| **48** | 1 | 1 | 1 | 1 | 1 | 1 |
| **468** | 2 | 1 | 1 | 2 | 2 | 1 |
| **155** | 1 | 2 | 1 | 1 | 2 | 2 |
| **1721** | 3 | 3 | 2 | 1 | 2 | 2 |
| **1208** | 4 | 3 | 3 | 1 | 2 | 2 |

Next steps: Generate code with X_train | View recommended plots | New interactive sheet

```
# import Random Forest classifier

from sklearn.ensemble import RandomForestClassifier

# instantiate the classifier

rfc = RandomForestClassifier(random_state=0)

# fit the model

rfc.fit(X_train, y_train)

# Predict the Test set results

y_pred = rfc.predict(X_test)


# Check accuracy score

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with 10 decision-trees : 0.9457
```

```
# instantiate the classifier with n_estimators = 100

rfc_100 = RandomForestClassifier(n_estimators=100, random_state=0)

# fit the model to the training set

rfc_100.fit(X_train, y_train)

# Predict on the test set results

y_pred_100 = rfc_100.predict(X_test)


# Check accuracy score
```

```
print('Model accuracy score with 100 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred_100)))
```

```
Model accuracy score with 100 decision-trees : 0.9457
```

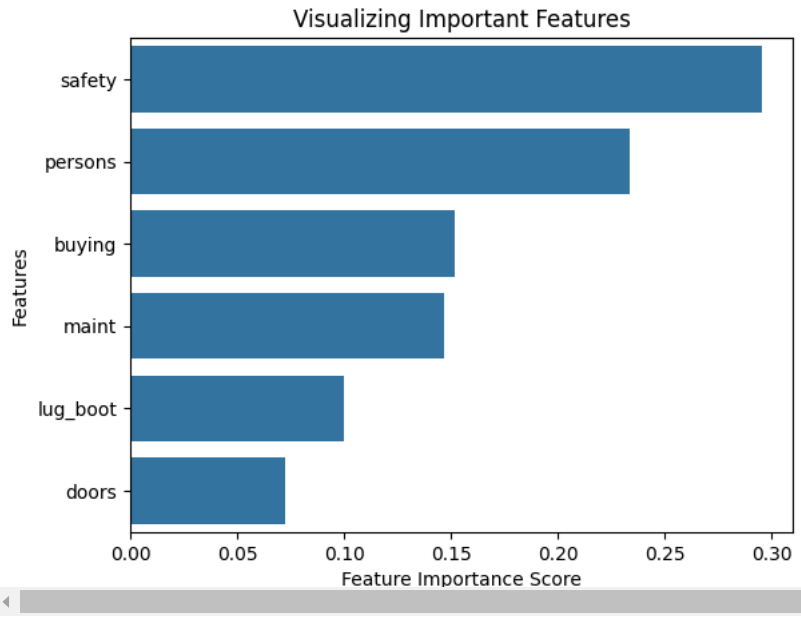
```
# create the classifier with n_estimators = 100

clf = RandomForestClassifier(n_estimators=100, random_state=0)


# fit the model to the training set

clf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).sort_values(ascending=False)

feature_scores
```

| | 0 |
|---|---|
| **safety** | 0.295319 |
| **persons** | 0.233856 |
| **buying** | 0.151734 |
| **maint** | 0.146653 |
| **lug_boot** | 0.100048 |
| **doors** | 0.072389 |

```
# Creating a seaborn bar plot

sns.barplot(x=feature_scores, y=feature_scores.index)

# Add labels to the graph

plt.xlabel('Feature Importance Score')

plt.ylabel('Features')

# Add title to the graph

plt.title("Visualizing Important Features")

# Visualize the graph

plt.show()
```



```
# declare feature vector and target variable

X = df.drop(['class', 'doors'], axis=1)

y = df['class']


# split data into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)


# encode categorical variables with ordinal encoding

encoder = ce.OrdinalEncoder(cols=['buying', 'maint', 'persons', 'lug_boot', 'safety'])


X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)


# instantiate the classifier with n_estimators = 100

clf = RandomForestClassifier(random_state=0)



# fit the model to the training set

clf.fit(X_train, y_train)


# Predict on the test set results

y_pred = clf.predict(X_test)



# Check accuracy score

print('Model accuracy score with doors variable removed : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with doors variable removed : 0.9264
```

Start coding or generate with AI.